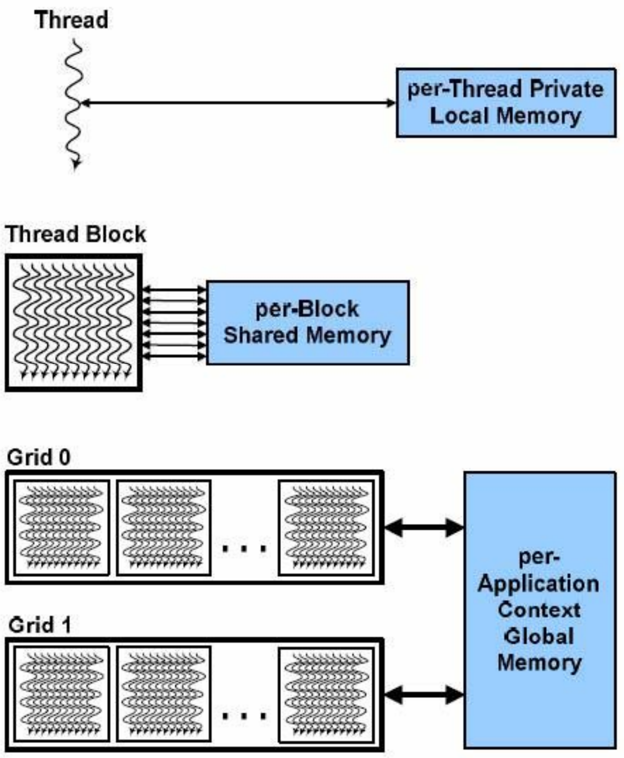
A Quick Refresher on CUDA

CUDA is the hardware and software architecture that enables NVIDIA GPUs to execute programs written with C, C++, Fortran, OpenCL, DirectCompute, and other languages. A CUDA program calls parallel kernels. A kernel executes in parallel across a set of parallel threads. The programmer or compiler organizes these threads in thread blocks and grids of thread blocks. The GPU instantiates a kernel program on a grid of parallel thread blocks. Each thread within a thread block executes an instance of the kernel, and has a thread ID within its thread block, program counter, registers, per-thread private memory, inputs, and output results.


Thread
per-Thread Private Local Memory
Thread Block
per-Block Shared Memory
Grid 0
Grid 1
per-Application Context Global Memory

CUDA Hierarchy of threads, blocks, and grids, with corresponding per-thread private, per-block shared, and per-application global memory spaces.

A thread block is a set of concurrently executing threads that can cooperate among themselves through barrier synchronization and shared memory. A thread block has a block ID within its grid.

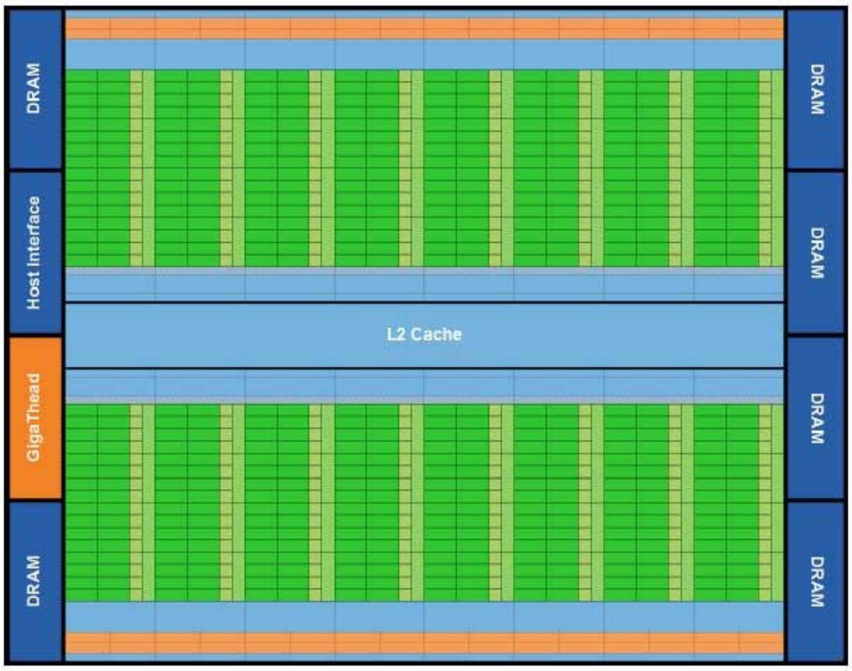
A grid is an array of thread blocks that execute the same kernel, read inputs from global memory, write results to global memory, and synchronize between dependent kernel calls. In the CUDA parallel programming model, each thread has a per-thread private memory space, used for register spills, function calls, and C automatic array variables. Each thread block has a per-Block shared memory space, used for inter-thread communication, data sharing, and result sharing in parallel algorithms. Grids of thread blocks share results in Global Memory space after kernel-wide global synchronization.

Hardware Execution

CUDA's hierarchy of threads maps to a hierarchy of processors on the GPU; a GPU executes one or more kernel grids; a streaming multiprocessor (SM) executes one or more thread blocks; and CUDA cores and other execution units in the SM execute threads. The SM executes threads in groups of 32 threads called a warp. While programmers can generally ignore warp execution for functional correctness and think of programming one thread, they can greatly improve performance by having threads in a warp execute the same code path and access memory in nearby addresses.

An Overview of the Fermi Architecture

The first Fermi based GPU, implemented with 3.0 billion transistors, features up to 512 CUDA cores. A CUDA core executes a floating point or integer instruction per clock for a thread. The 512 CUDA cores are organized in 16 SMs of 32 cores each. The GPU has six 64-bit memory partitions, for a 384-bit memory interface, supporting up to a total of 6 GB of GDDR5 DRAM memory. A host interface connects the GPU to the CPU via PCI-Express. The GigaThread global scheduler distributes thread blocks to SM thread schedulers.

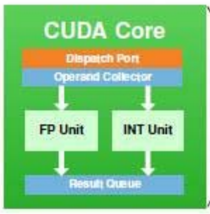


Fermi's 16 SM are positioned around a common L2 cache. Each SM is a vertical rectangular strip that contain an orange portion (scheduler and dispatch), a green portion (execution units), and light blue portions (register file and L1 cache).

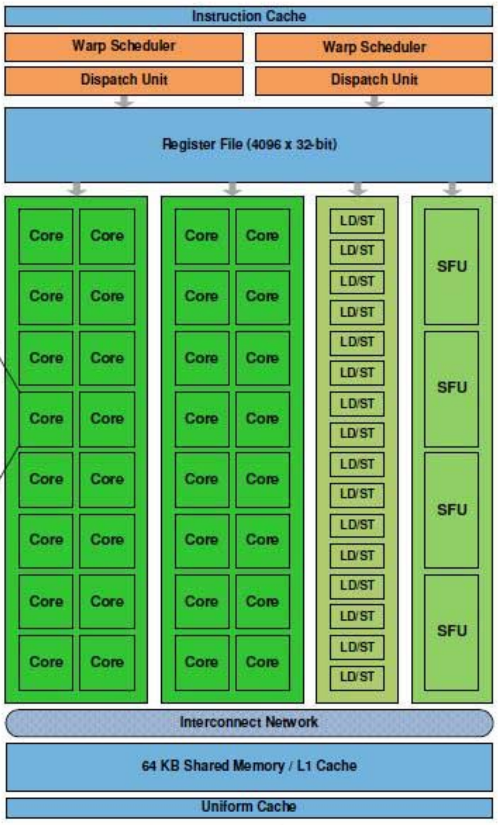
Third Generation Streaming Multiprocessor

The third generation SM introduces several architectural innovations that make it not only the most powerful SM yet built, but also the most programmable and efficient.

512 High Performance CUDA cores



Each SM features 32 CUDA processors—a fourfold increase over prior SM designs. Each CUDA processor has a fully pipelined integer arithmetic logic unit (ALU) and floating point unit (FPU). Prior GPUs used IEEE 754-1985 floating point arithmetic. The Fermi architecture implements the new IEEE 754-2008 floating-point standard, providing the fused multiply-add (FMA) instruction for both single and double precision arithmetic. FMA improves over a multiply-add (MAD) instruction by doing the multiplication and addition with a single final rounding step, with no loss of precision in the addition. FMA is more accurate than performing the operations separately. GT200 implemented double precision FMA.



Fermi Streaming Multiprocessor (SM)

In GT200, the integer ALU was limited to 24-bit precision for multiply operations; as a result, multi-instruction emulation sequences were required for integer arithmetic. In Fermi, the newly designed integer ALU supports full 32-bit precision for all instructions, consistent with standard programming language requirements. The ALU is optimized to efficiently support 64-bit and extended precision operations. Various instructions are supported, including Boolean, shift, move, compare, convert, bit-field extract, bit-reverse insert, and population count.

16 Load/Store Units

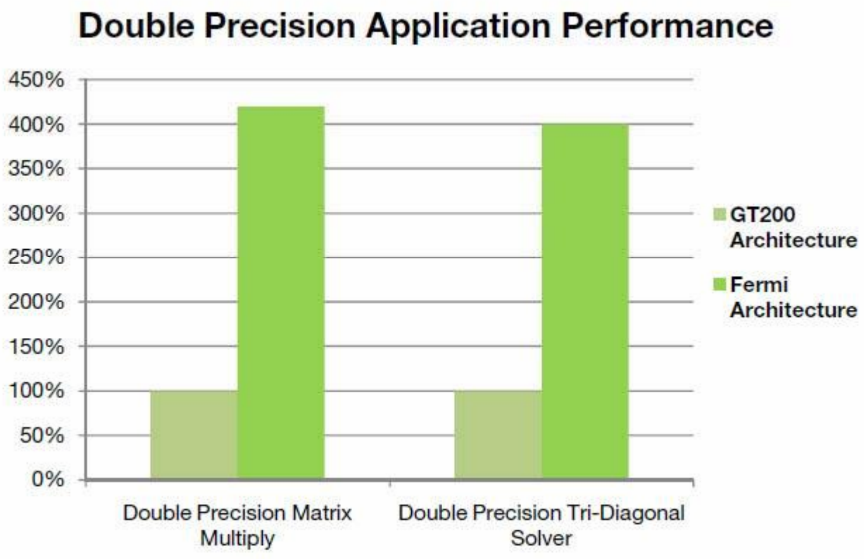
Each SM has 16 load/store units, allowing source and destination addresses to be calculated for sixteen threads per clock. Supporting units load and store the data at each address to cache or DRAM.

Four Special Function Units

Special Function Units (SFUs) execute transcendental instructions such as sin, cosine, reciprocal, and square root. Each SFU executes one instruction per thread, per clock; a warp executes over eight clocks. The SFU pipeline is decoupled from the dispatch unit, allowing the dispatch unit to issue to other execution units while the SFU is occupied.

Designed for Double Precision

Double precision arithmetic is at the heart of HPC applications such as linear algebra, numerical simulation, and quantum chemistry. The Fermi architecture has been specifically designed to offer unprecedented performance in double precision; up to 16 double precision fused multiply-add operations can be performed per SM, per clock, a dramatic improvement over the GT200 architecture.



Early performance evaluations show Fermi performing up to 4.2x faster than GT200 in double precision applications.